CMPSC-265
Data Structures and Algorithms

Zaihan Yang
zyang13@suffolk.edu

Department of Math and Computer Science
Suffolk University

Fall 2019

Recap

- Efficiency of Stack, Queue and Priority Queue
- The linked list data structure:
  - Singly linked list
    - Single-ended singly linked list
    - Main operations on single-ended singly linked list and its time complexity.

Learning Topics

- Time complexity analysis for the basic operations of single-ended singly linked list
- Applications on Linked list
- Double-ended singly linked list
- Implementation of Stack and Queue using Linked list
- Doubly linked list

Single-ended Singly Linked List Operations' Time Complexity

Operations	Time Complexity
Insert at First	O(1)
Insert at End	O(N)
Delete from First	O(1)
Delete from End	O(N)
Traverse	O(N)
Search for a specific node	O(N)
Delete a specific node	O(N)

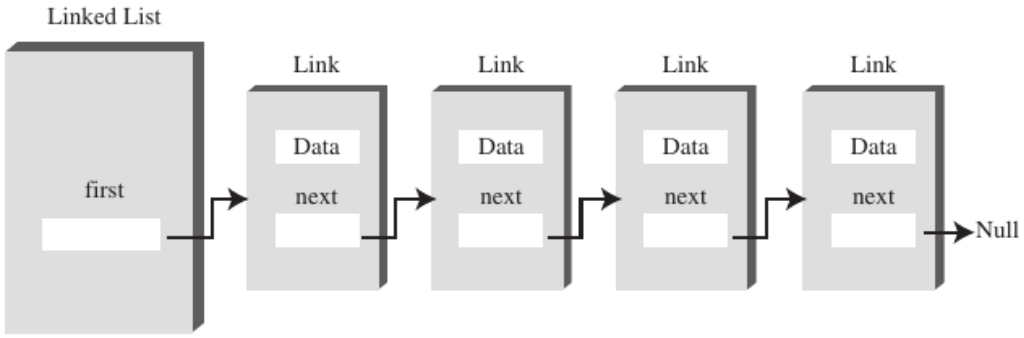
- Faster to insert and delete compared to arrays, since the items do not need to be shifted
- You can expand the list dynamically, memory is allocated on demand.

Linked list Applications

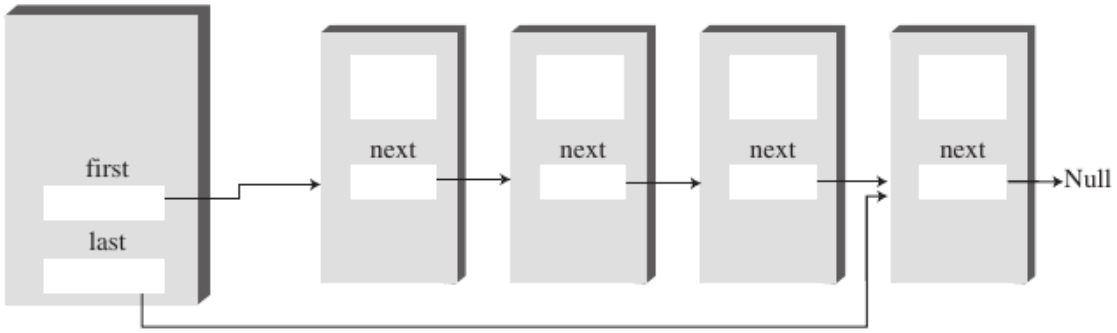
- Problem:
- Reverse a singly linked list:
- Example:
  - Input: 1->2->3->4->5->null
  - Output: 5->4->3->2->1->null

Double-ended Singly Linked List

- Linked List



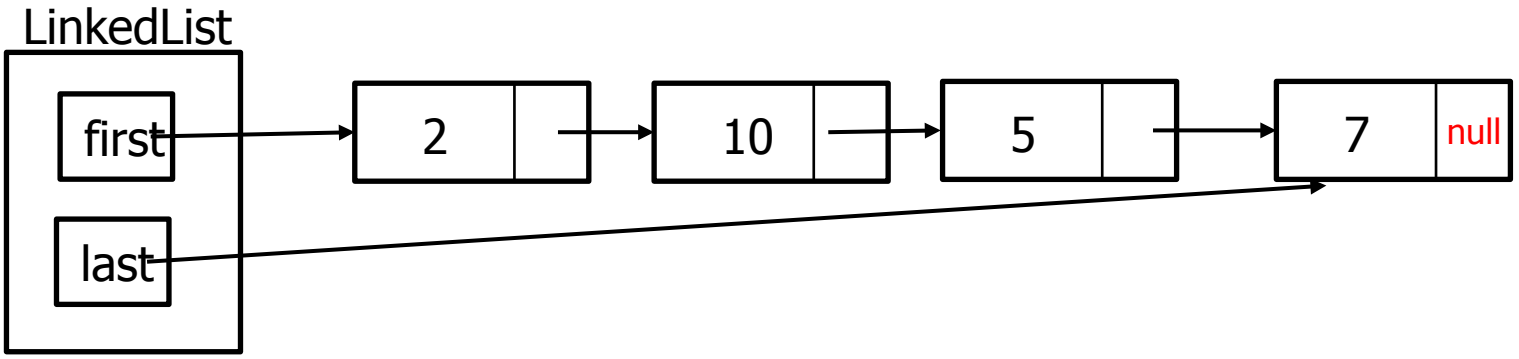
- Double-ended Linked List



Robert Lafore. 2002. Data Structures and Algorithms in Java (2 ed.). Sams, Indianapolis, IN, USA. Page 180

Double-ended LinkedList

- It also keeps a reference to the last Link.



- We can insert at the end without traversing the list
  - Good for implementing Queue
- How about deleting the last Link?

The Double-ended LinkedList Class

```
class DoubleEndedLinkedList
{
        private Link first; // ref to the first link on list
        private Link last; // ref to the last link on list
// -----------------------------------------------------------
        public void DoubleEndedLinkedList() // constructor
        {
                first = null;
                last = null;
        }
// -----------------------------------------------------------
        // these methods are similar to the LinkedList class
        public boolean isEmpty() {}
        public Link deleteFirst() {}
        public void displayList() {}
```

The Double-ended LinkedList Class

```
public void insertFirst(int data)
 {
        Link newLink = new Link(data); // create a new link
        if(isEmpty())
                last=newLink;
        newLink.next = first;
        first = newLink;
}
// ----------------------------------------------------------
public void insertLast(int data)
 {
        Link newLink = new Link(data); // create a new link
        if(isEmpty())
                first=newLink;
        else
                last.next = newLink;
        last = newLink;
}
}//end of class
```

Double-ended LinkedList Demo

```
class DELLDemo
{
        public static void main(String[] args)
        {
                DoubleEndedLinkedList myList = new DoubleEndedLinkedList();

                myList.insertFirst(3);
                myList.insertFirst (2);
                myList.insertFirst (1);

                myList.insertLast(1);
                myList.insertLast (2);
                myList.insertLast (3);

                myList.displayList();

        }
}
```

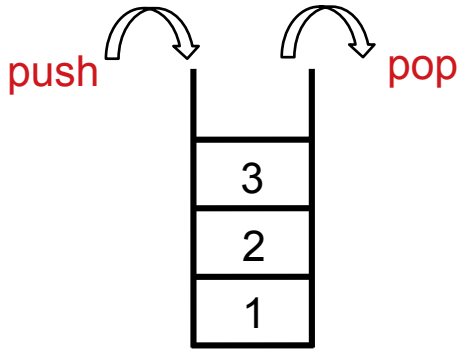
Double-ended Singly Linked List Operations' Time Complexity

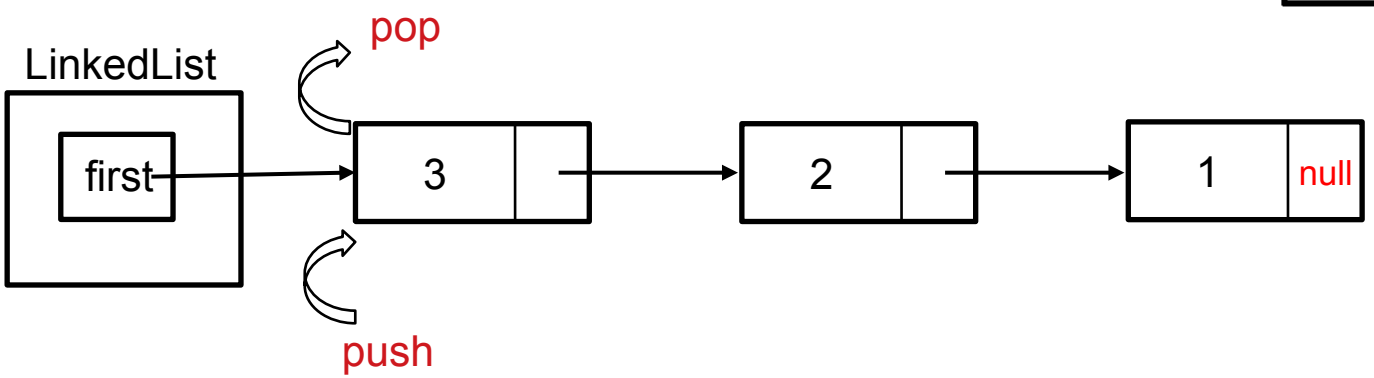
Operations	Time Complexity
Insert at First	O(1)
Insert at End	O(1)
Delete from First	O(1)
Delete from End	O(1)
Traverse	O(N)
Search for a specific node	O(N)
Delete a specific node	O(N)

- Faster to insert and delete compared to arrays, since the items do not need to be shifted
- You can expand the list dynamically, memory is allocated on demand.

Stack Implementation using single-ended singly Linked List

- No need to specify maxSize/capacity
- Push() → insertFirst()
- pop() → deleteFirst()



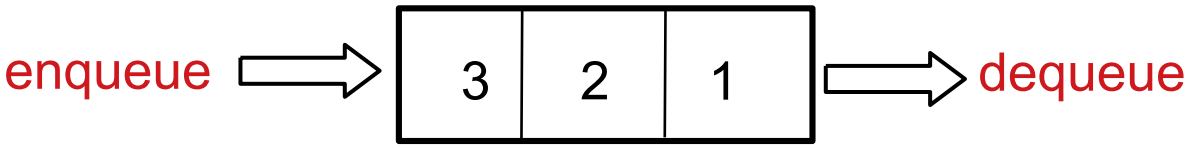


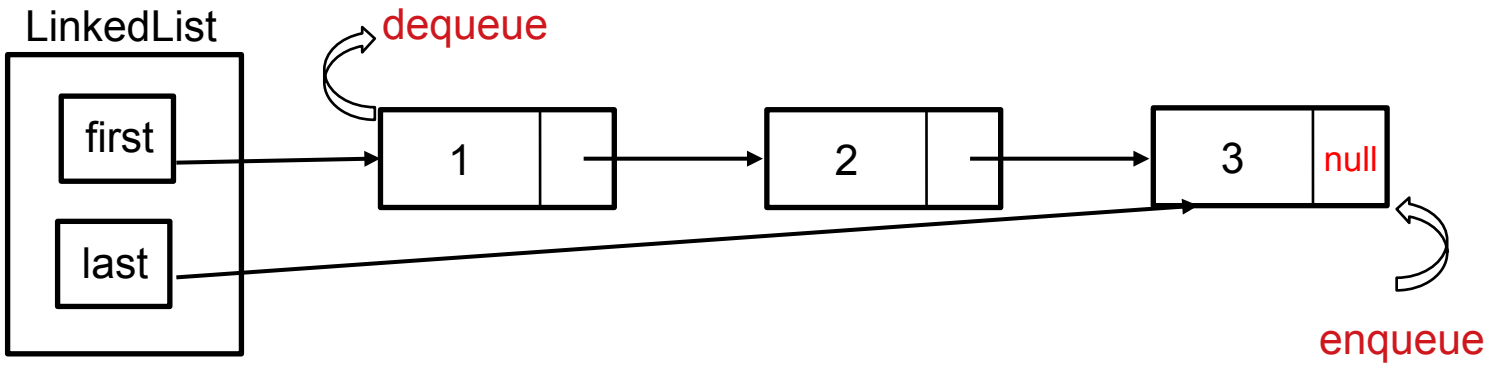
Linked List-based Stack Class

```
class LinkStack {
        private LinkedList theList;

        // constructor
        public LinkStack()
        { theList = new LinkedList(); }

        public boolean isEmpty()
        {return theList.isEmpty();}

        public void push(int item) {
                theList.insertFirst(item);
                }
        public int pop() {
                return theList.deleteFirst();
                }
        public void displayStack() {
                theList.displayList();
                }}
```

Queue Implementation using Double-ended Singly Linked List

- No need to specify maxSize/capacity
- enqueue() →insertLast()
- dequeue() → deleteFirst()





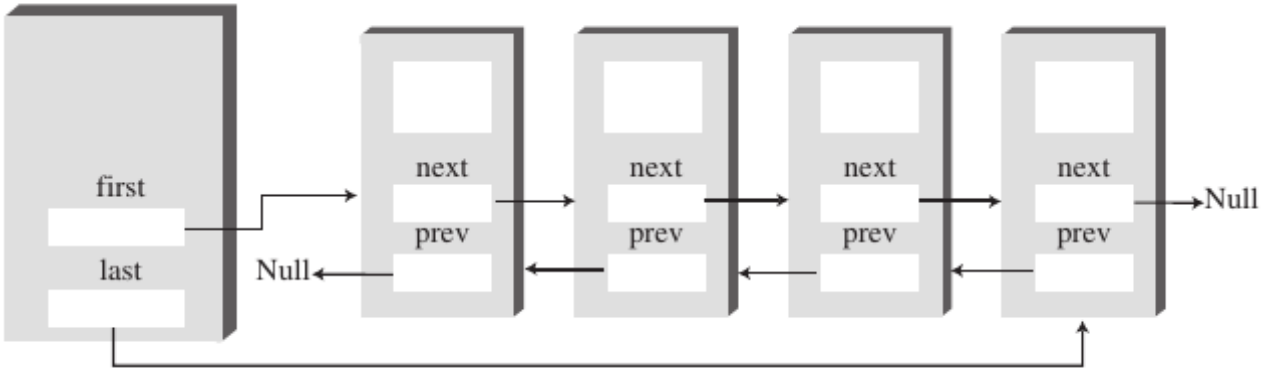
Linked List-based Queue Class

```
class LinkQueue {
        private DoubleEndedLinkedList theList;

        // constructor
        public LinkQueue()
        { theList = new DoubleEndedLinkedList(); }

        public boolean isEmpty()
        {return theList.isEmpty();}

        public void enqueue(int item) {
                theList.insertLast(item);
                }
        public int dequeue() {
                return theList.deleteFirst();
                }
        public void displayQueue() {
                theList.displayList();
                }}
```

Doubly Linked List

- Two references at each Link(node)
  - next: reference to the next Link
  - prev: reference to the previous Link

- Allows going forward or backward on the list



Robert Lafore. 2002. Data Structures and Algorithms in Java (2 ed.). Sams, Indianapolis, IN, USA. Page 222

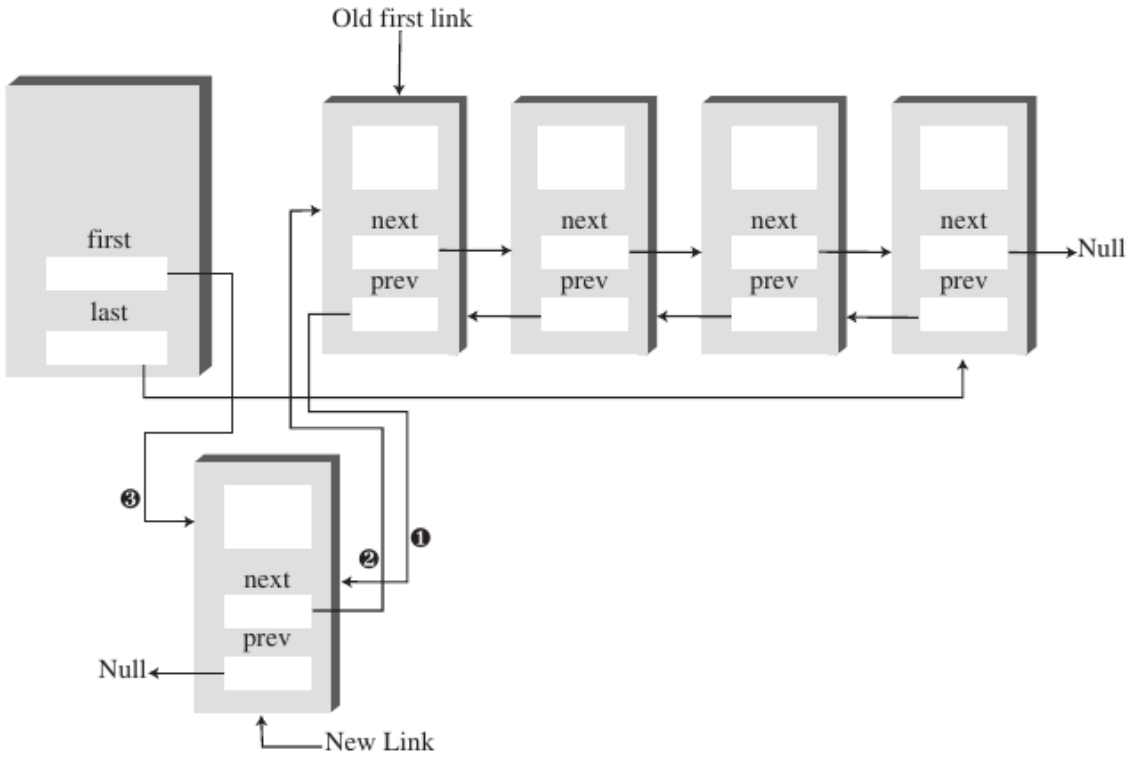
Doubly Linked List- Link Class

```
class Link
{
        public int data; // data
        public Link next; // reference to next link
        public Link prev; // reference to previous link

        // Constructor
        public Link(int data)
        {
                this.data=data;
                next = null;
                prev = null;
        }
}
```

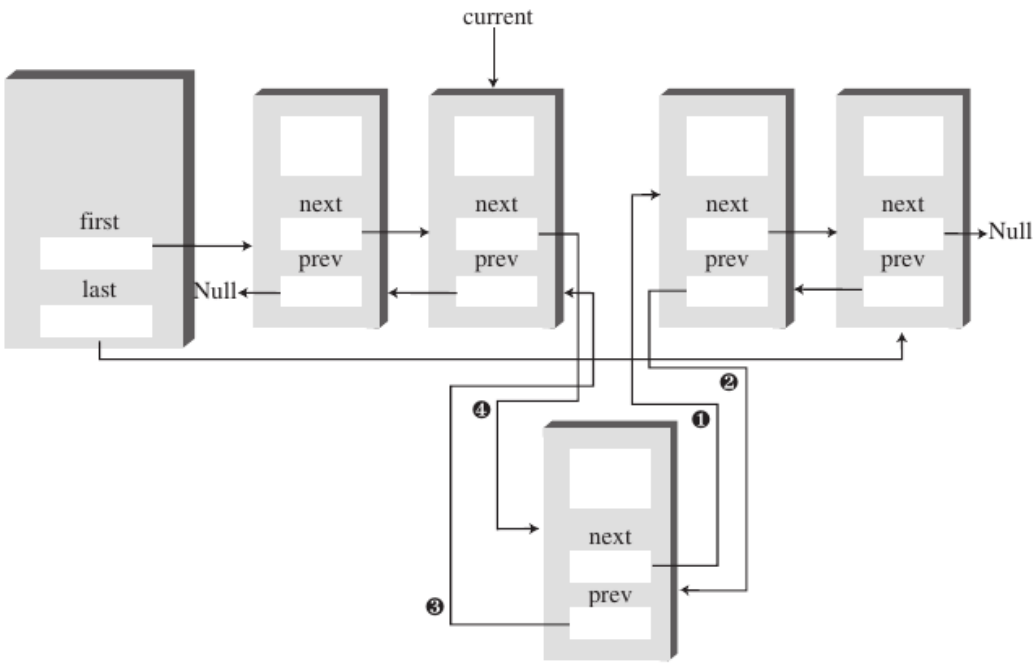
Doubly Linked List Class

```
class DoublyLinkedList
{
        private Link first;// Reference to the first link
        private Link last; // Reference to the last link

        // constructor
        public DoublyLinkedList()
        { first = null; last= null;}

        public void displayListForward() {
                Link current = first;
                while (current!=null) {
                        System.out.println(current.data);
                        current = current.next; }}

        public void displayListBackward() {
                Link current = last;
                while (current!=null) {
                        System.out.println(current.data);
                        current = current.prev; }}
```

InsertFirst in a Doubly Linked List

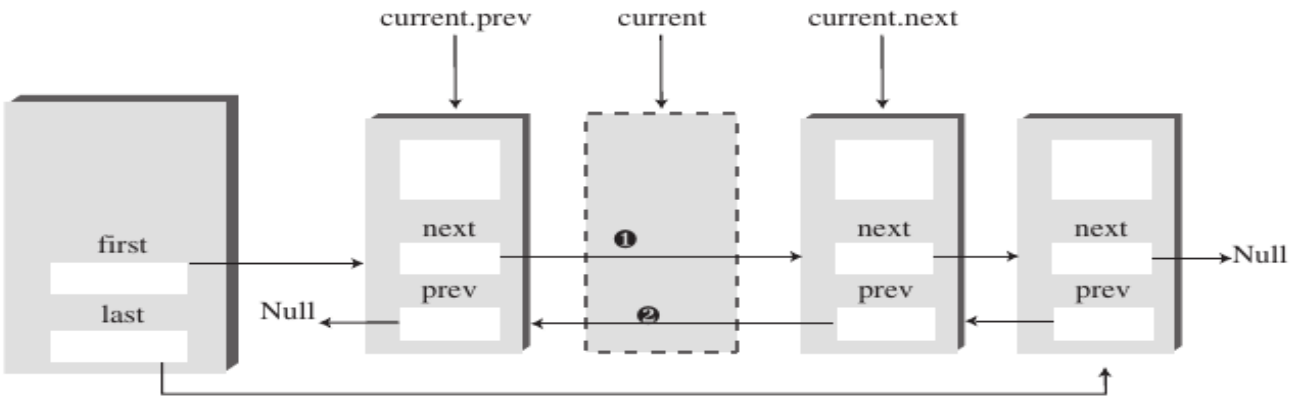


InsertAfter in a Doubly Linked List



Robert Lafore. 2002. Data Structures and Algorithms in Java (2 ed.). Sams, Indianapolis, IN, USA. Page 224

Delete in a Doubly Linked List



Doubly Linked List

- Can traverse backwards
- Can delete at the end in O(1)
- Two references need to be updated

Operations	Time Complexity
Insert at First	O(1)
Insert at End	O(1)
Delete from First	O(1)
Delete from End	O(1)
Traverse	O(N)
Search for a specific node	O(N)
insertAfter a specific node	O(N)
Delete a specific node	O(N)